GFF3 Toolkit Documentation

NAL-i5K

Nov 14, 2018

Contents:

CHAPTER 1

gff3_QC readme

1.1 Usage

gff3_QC.py [-h] [-g GFF] [-f FASTA] [-noncg] [-i] [-n ALLOWED_NUM_OF_N][-t [CHECK_N_FEATURE_TYPES [CHECK_N_FEATURE_TYPES ...]]] [-o OUTPUT] [-v] [-s STATISTIC]

1.2 Testing environment

Python 2.7

1.3 Inputs

1. GFF3: Specify the file name with the -g or –gff argument. Please note that this program requires gene/pseudogene and mRNA/pseudogenic_transcript to have an ID attribute in column 9.
2. Fasta file: Specify the file name with the -f or –fasta argument. This file **must** be the Fasta file that the GFF3 seqids and coordinates refer to. For more information, refer to the GFF3 specification.

1.4 Outputs

1. Error report for the input GFF3 file
    - Line_num: Line numbers of the found problematic models in the input GFF3 file.
    - Error_code: Error codes for the found problematic models. Please refer to lib/ERROR/ERROR.py to see the full list of Error_code and the corresponding Error_tag.
        - Error_tag: Detail of the found errors for the problematic models. Please refer to lib/ERROR/ERROR.py to see the full list of Error_code and the corresponding Error_tag.

2. Statistic report for the output files
    - Error_code: Error codes for the found problematic models. Please refer to lib/ERROR/ERROR.py to see the full list of Error_code and the corresponding Error_tag.
    - Number of problematic models: Calculate the type and number of error_code.
        - Error_tag: Detail of the found errors for the problematic models. Please refer to lib/ERROR/ERROR.py to see the full list of Error_code and the corresponding Error_tag.

1.5 Quick start

```
gff3_QC -g example_file/example.gff3 -f example_file/reference.fa -o test -s
statistic.txt
```

or

```
gff3_QC --gff example_file/example.gff3 --fasta example_file/reference.fa
--output test --statistic statistic.txt
```

1.6 Optional arguments

1. -h, –help
    - show this help message and exit
2. -g GFF, –gff GFF
    - Genome annotation file, gff3 format
3. -f FASTA, –fasta FASTA
    - Genome sequences, fasta format
4. -noncg, –noncanonical_gene
    - gff3 file is not formatted in the canonical gene model format.
5. -i, –initial_phase
    - Check whether initial CDS phase is 0 (default - no check)
6. -n ALLOWED_NUM_OF_N, –allowed_num_of_n ALLOWED_NUM_OF_N
    - Max number of Ns allowed in a feature, anything more will be reported as an error (default: 0)
7. -t [CHECK_N_FEATURE_TYPES [CHECK_N_FEATURE_TYPES ...]], –check_n_feature_types [CHECK_N_FEATURE_TYPES [CHECK_N_FEATURE_TYPES ...]]
    - Count the number of Ns in each feature with the type specified, multiple types may be specified, ex: -t CDS exon (default: “CDS”)
8. -o OUTPUT, –output OUTPUT
    - output file name (default: report.txt)
9. -s STATISTIC, –statistic STATISTIC
    - statistic file name (default: statistic.txt
10. -v, –version
    - show program’s version number and exit

1.7 More information

- gff3_QC.py full documentation

CHAPTER 2

gff3_QC full documentation

2.1 Background

The GFF3 format is flexible and easy to use for most biologists, but this flexibility also allows many errors to be introduced. This QC program aims to detect over 50 types of formatting errors.

Errors are detected by reviewing three types of feature sets in a GFF3 file, and thus are grouped into three categories (Error category – feature type):

- Intra-model errors (Ema) – multiple features within a model
- Inter-model errors (Emr) – multiple features across models
- Single feature errors (Esf) – each single feature.

In addition, we distinguish between errors that apply to protein-coding genes in the 'canonical' Sequence ontology style, and errors that apply to 'non-canonical' gene models – i.e. non-coding models, or protein-coding genes that are not modeled with gene, mRNA, CDS and exon features. To perform error-checking on a gff3 file that contains non-canonical gene models, you can specify the –noncg argument when running the program.

Below we list all errors currently considered by gff3_QC.py, including the error code, the error tag (a brief explanation of the error), and whether the error is checked for non-canonical gene models (when using the –noncg argument).

View the gff3_QC.py readme for instructions on how to run the program.

2.1.1 Intra-model: Multiple features within a model (Ema)

The error category 'Intra-model' collects formatting errors that can be found by jointly considering multiple features within a gene model, such as gene, mRNA, exon, and CDS features. Errors in this category are given an 'Error_Code' starting with 'Ema'.

|Error_Code|Error_Tag|Checked if non-canonical| |:——|:——|:——| |Ema0001|Parent feature start and end coordinates exceed those of child features|Yes| |Ema0002|Protein sequence contains internal stop codons|No| |Ema0003|This feature is not contained within the parent feature coordinates|Yes| |Ema0004|Incomplete gene feature that should contain at least one mRNA, exon, and CDS|No| |Ema0005|Pseudogene has invalid child feature type|Yes| |Ema0006|Wrong

phase|No| |Ema0007|CDS and parent feature on different strands|Yes| |Ema0008|Warning for distinct isoforms that do not share any regions|No| |Ema0009|Incorrectly merged gene parent? Isoforms that do not share coding sequences are found|No|

2.1.2 Inter-model: Multiple features across models (Emr)

The error category 'Inter-model' collects formatting errors that can be found by comparing multiple gene models. Errors in this category are given an 'Error_Code' starting with 'Emr'.

|Error_Code|Error_Tag|Checked if non-canonical| |:——|:——|:——| |Emr0001|Duplicate transcript found|No| |Emr0002|Incorrectly split gene parent?|No| |Emr0003|Duplicate ID|Yes|

2.1.3 Single feature (Esf)

The error category 'Single Feature' collects formatting errors that can be found by searching the GFF3 file line by line. Errors in this category are given an 'Error_Code' starting with 'Esf'.

|Error_Code|Error_Tag|Checked if non-canonical| |:——|:——|:——| |Esf0001|Feature type may need to be changed to pseudogene|Yes| |Esf0002|Start/Stop is not a valid 1-based integer coordinate|Yes| |Esf0003|strand information missing|Yes| |Esf0004|Seqid not found in any ##sequence-region|Yes| |Esf0005|Start is less than the ##sequence-region start|Yes| |Esf0006|End is greater than the ##sequence-region end|Yes| |Esf0007|Seqid not found in the embedded ##FASTA|Yes| |Esf0008|End is greater than the embedded ##FASTA sequence length|Yes| |Esf0009|Found Ns in a feature using the embedded ##FASTA|Yes| |Esf0010|Seqid not found in the external FASTA file|Yes| |Esf0011|End is greater than the external FASTA sequence length|Yes| |Esf0012|Found Ns in a feature using the external FASTA|Yes| |Esf0013|White chars not allowed at the start of a line|Yes| |Esf0014|##gff-version" missing from the first line|Yes| |Esf0015|Expecting certain fields in the feature|Yes| |Esf0016|##sequence-region seqid may only appear once|Yes| |Esf0017|Start/End is not a valid integer|Yes| |Esf0018|Start is not less than or equal to end|Yes| |Esf0019|Version is not "3"|Yes| |Esf0020|Version is not a valid integer|Yes| |Esf0021|Unknown directive|Yes| |Esf0022|Features should contain 9 fields|Yes| |Esf0023|escape certain characters|Yes| |Esf0024|Score is not a valid floating point number|Yes| |Esf0025|Strand has illegal characters|Yes| |Esf0026|Phase is not 0, 1, or 2, or not a valid integer|Yes| |Esf0027|Phase is required for all CDS features|Yes| |Esf0028|Attributes must escape the percent (%) sign and any control characters|Yes| |Esf0029|Attributes must contain one and only one equal (=) sign|Yes| |Esf0030|Empty attribute tag|Yes| |Esf0031|Empty attribute value|Yes| |Esf0032|Found multiple attribute tags|Yes| |Esf0033|Found ", " in a attribute, possible unescaped|Yes| |Esf0034|attribute has identical values (count, value)|Yes| |Esf0035|attribute has unresolved forward reference|Yes| |Esf0036|Value of a attribute contains unescaped ","|Yes| |Esf0037|Target attribute should have 3 or 4 values|Yes| |Esf0038|Start/End value of Target attribute is not a valid integer coordinate|Yes| |Esf0039|Strand value of Target attribute has illegal characters|Yes| |Esf0040|Value of Is_circular attribute is not "true"|Yes| |Esf0041|Unknown reserved (uppercase) attribute|Yes|

CHAPTER 3

gff3_fix readme

3.1 Usage

gff3_fix.py [-h] [-qc_r QC_REPORT] [-g GFF] [-og OUTPUT_GFF] [-v]

3.2 Testing environment

Python 2.7

3.3 Inputs

1. Error report: Error report from gff3_QC.py. Specify the file name with the -qc_r or –qc_report argument. Error report should only include those errors that should be fixed. If errors identified by gff3_QC.py should not be fixed, remove lines containing errors from report file.
2. GFF3: Specify the file name with the -g or –gff argument.

3.4 Outputs

1. Corrected GFF3

3.5 Quick start

```
gff3_fix -qc_r error.txt -g example_file/example.gff3 -og corrected.gff3
```

3.6 Optional arguments

1. -h, –help
    - show this help message and exit
2. -qc_r QC_REPORT, –qc_report QC_REPORT
    - Error report from gff3_QC.py
3. -g GFF, –gff GFF
    - Genome annotation file, gff3 format
4. -og OUTPUT_GFF, –output_gff OUTPUT_GFF
    - output gff3 file name (default: corrected.gff3)
5. -v, –version
    - show program's version number and exit

3.7 More information

- gff3_fix.py full documentation

CHAPTER 4

gff3_fix full documentation

4.1 Background

The gff3_fix program fixes 30 error types detected by the program gff3_QC.py. The section 'gff3_fix' lists all error types that currently can be fixed by the gff3_fix.py function (currently 30), including the method used for the fix. (Note that in some cases, this means removing the affected gene model). The section 'Fix function' describes the methods used to fix the error type in question. The section 'Currently no automatic fix available' lists the error types which gff3_fix currently does not handle.

4.2 gff3_fix

|Error code|Error tag|Fix function| |—|—|—| Ema0001|Parent feature start and end coordinates exceed those of child features|fix_boundary Ema0003|This feature is not contained within the parent feature coordinates|fix_boundary Ema0005|Pseudogene has invalid child feature type|pseudogene Ema0006|Wrong phase|fix_phase Ema0007|CDS and parent feature on different strands|delete_model Ema0009|Incorrectly merged gene parent? Isoforms that do not share coding sequences are found|split Emr0001|Duplicate transcript found|remove_duplicate_trans Emr0002|Incorrectly split gene parent?|merge Esf0001|Feature type may need to be changed to pseudogene|pseudogene Esf0002|Start/Stop is not a valid 1-based integer coordinate|delete_model Esf0003|strand information missing|delete_model Esf0013|White chars not allowed at the start of a line|gff3 parse Esf0014|##gff-version" missing from the first line|add_gff3_version Esf0016|##sequence-region seqid may only appear once|remove_directive Esf0017|Start/End is not a valid integer|delete_model Esf0018|Start is not less than or equal to end|delete_model Esf0020|Version is not a valid integer|remove_directive Esf0021|Unknown directive|remove_directive Esf0022|Features should contain 9 fields|delete_model Esf0025|Strand has illegal characters|delete_model Esf0026|Phase is not 0, 1, or 2, or not a valid integer|fix_phase Esf0027|Phase is required for all CDS features|fix_phase Esf0029|Attributes must contain one and only one equal (=) sign|fix_attributes Esf0030|Empty attribute tag|fix_attributes Esf0031|Empty attribute value|fix_attributes Esf0032|Found multiple attribute tags|fix_attributes Esf0033|Found ", " in a attribute, possible unescaped|fix_attributes Esf0034|attribute has identical values (count, value)|fix_attributes Esf0036|Value of a attribute contains unescaped ","|fix_attributes Esf0041|Unknown reserved (uppercase) attribute|fix_attributes Esf0041|Unknown reserved (uppercase) attribute|fix_attributes

4.3 Fix function

|fix function|method| |—|—| |delete_model|remove the whole model from the original gff3 file| |remove_duplicate_trans|remove the duplicate transcripts| |remove_directive|remove the directive| |pseudogene|remove CDS feature and change the feature type of the other feature: first-level → pseudogene; second-level → pseudogenic_transcript; third-level(exon) → pseudogenic_exon| |fix_boundary|update the coordinate of the parent by using the minimum and the maximum coordinate of the child feature| |fix_phase|correct phase by the function `next_phase = (3 - ((CDS['end'] - CDS['start'] + 1 - phase) % 3)) % 3`. Note: If the first CDS segment doesn't have a phase, the initial phase will be 0.| |fix_attributes|remove empty attribute tag/value; remove the redundant equal sign(=); remove dupliacte attribute; make the first character of the unknown reserved attribute lower case; merge multiple attribute tag and remove the duplicate attribute value; replace `,` with `%2C`| |split|split the incorrectly merged transcript from a gene model and generate a new gene model| |merge|merge the incorrectly split gene model| |add_gff3_version|Add `##gff-version 3` to the first line of gff3 file| |gff3 parse|parse the gff3 file; ignore blank line in gff3; remove the white chars at the start of a line|

4.4 Currently no automatic fix available

|Error code|Error tag| |—|—| Ema0002 |Protein sequence contains internal stop codons| Ema0004 |Incomplete gene feature that should contain at least one mRNA, exon, and CDS| Ema0008 |Warning for distinct isoforms that do not share any regions| Emr0003 |Duplicate ID| Esf0004 |Seqid not found in any ##sequence-region| Esf0005 |Start is less than the ##sequence-region start| Esf0006 |End is greater than the ##sequence-region end| Esf0007 |Seqid not found in the embedded ##FASTA| Esf0008 |End is greater than the embedded ##FASTA sequence length| Esf0009 |Found Ns in a feature using the embedded ##FASTA| Esf0010 |Seqid not found in the external FASTA file| Esf0011 |End is greater than the external FASTA sequence length| Esf0012 |Found Ns in a feature using the external FASTA| Esf0015 |Expecting certain fields in the feature| Esf0019 |Version is not "3"| Esf0023 |escape certain characters| Esf0024 |Score is not a valid floating point number| Esf0035 |attribute has unresolved forward reference| Esf0037 |Target attribute should have 3 or 4 values| Esf0038 |Start/End value of Target attribute is not a valid integer coordinate| Esf0039 |Strand value of Target attribute has illegal characters| Esf0040 |Value of Is_circular attribute is not "true"|

CHAPTER 5

gff3_merge readme

5.1 Usage

gff3_merge.py [-h] [-g1 GFF_FILE1] [-g2 GFF_FILE2] [-f FASTA] [-u1 USER_DEFINED_FILE1] [-u2 USER_DEFINED_FILE2] [-og OUTPUT_GFF] [-r REPORT_FILE] [-a] [-noAuto] [-v]

5.2 Testing environment

1. Python 2.7
2. Perl v5.16.3

5.3 Inputs

1. GFF3 file with new or modified annotations, to be merged into GFF3 file 2. Specify the file name with the -g1 or –gff_file1 argument. Please note that this program requires gene/pseudogene and mRNA/pseudogenic_transcript to have an ID attribute in column 9. If replace tags are present (see below), these tags **must** refer to transcript/mRNA model IDs in the reference GFF3 file, specified by -g2.
2. Reference models in GFF3 format: Specify the file name with the -g2 or –gff_file2 argument. The models from -g1 will be merged into this file, replacing models in -g2. Please note that this program requires gene/pseudogene and mRNA/pseudogenic_transcript to have an ID attribute in column 9. If the reference GFF3 file contains gene models with multiple isoforms, please review the section "Odd use cases" below prior to running the program.
3. Fasta file: Specify the file name with the -f or –fasta argument. This file **must** be the Fasta file that the GFF3 seqids and coordinates in both GFF3 files refer to. For more information, refer to the GFF3 specification.

5.4 Outputs

1. .gff: A merged gff3 file
2. .txt: Merge log file

5.5 Quick start

- Merge the two files with auto-assignment of replace tags (default) `gff3_merge -g1 example_file/new_models.gff3 -g2 example_file/reference.gff3 -f example_file/reference.fa -og merged.gff -r merged_report.txt`
- If your GFF3 files have proper replace tags at column 9 (Format: replace=[Transcript ID]), you can merge the two GFF3 files without auto-assignment of replace tags. `gff3_merge -g1 example_file/new_models_w_replace.gff3 -g2 example_file/reference.gff3 -f example_file/reference.fa -og merged.gff -r merged_report.txt -noAuto`

5.6 Optional arguments

1. -h, –help
   - show this help message and exit
2. -g1 GFF_FILE1, –gff_file1 GFF_FILE1
   - Updated GFF3 file, such as Apollo gff
3. -g2 GFF_FILE2, –gff_file2 GFF_FILE2
   - Reference GFF3 file, such as Maker gff or OGS gff
4. -f FASTA, –fasta FASTA
   - Genomic sequences in the fasta format
5. -u1 USER_DEFINED_FILE1, –user_defined_file1 USER_DEFINED_FILE1
   - File for specifing parent and child features for fasta extraction from updated GFF3 file.
6. -u2 USER_DEFINED_FILE2, –user_defined_file2 USER_DEFINED_FILE2
   - File for specifing parent and child features for fasta extraction from reference GFF3 file.
7. -og OUTPUT_GFF, –output_gff OUTPUT_GFF
   - The merged GFF3 file (default: merged.gff)
8. -r REPORT_FILE, –report_file REPORT_FILE
   - Log file for the integration (default: merge_report.txt)
9. -a, –all
   - auto-assignment replace tags for all transcript features. (default: Only automatically assign replace tags for the transcript without replace tags)
10. -noAuto, –auto_assignment
    - Turn off the auto-assignment of replace tags, if you have had the replace tags in your update gff (default: Automatically assign replace tags and then merge the gff files)

11. -v, –version
    - show program's version number and exit

5.7 More information

- gff3_merge.py full documentation

CHAPTER 6

gff3_merge full documentation

6.1 Table of Contents

6.2 Background

The program gff3_merge.py was developed to merge output from the manual annotation program Apollo (http://genomearchitect.github.io/) with a single reference GFF3 file as part of the i5k pilot project. The idea is to have a program that will take manual annotations from Apollo, and fold these into a single reference gene set, where manual annotations replace overlapping models in the reference gene set.

At a minimum, we recommend running the program gff3_QC.py on the manual annotation GFF3 prior to running gff3_merge.py, if not also the reference GFF3 file. Otherwise, you may incorporate errors into the merged GFF3 file, or the merge program may not work to begin with.

The program gff3_merge.py can be conceptually separated into 3 steps:

1. Recognize or auto-assign *Replace Tags* to transcripts or mRNAs in the modified GFF3 file
2. Determine merge actions based on the Replace Tags:

- deletion – a model has the status 'Delete'

- simple replacement – a model has a single replace tag
- new addition – a model has a replace tag 'NA'
- split replacement – a modified model shares a replace tag with other modified models
- merge replacement – a model has multiple replace tags

1. Models from modified GFF3 file replace models from reference GFF3 file based on merge actions in step 2.

Note that all information, including functional information (e.g. Name, Dbxrefs, etc.), from the modified GFF3 file replaces the corresponding reference information in the merged GFF3 file, meaning that any functional information in models slated to be replaced in the reference GFF3 file will NOT be carried over into the merged GFF3 file.

View the gff3_merge.py readme for instructions on how to run the program.

6.3 Replace Tags

(*back*)

The replace tag is a custom GFF3 attribute in the new or modified GFF3 file that specifies which mRNA(s) or transcript(s) from a single reference GFF3 file should be replaced by the new annotation. The replace tag follows this format: replace=[Name or ID attribute of reference mRNA or transcript to be replaced]. The replace tag could be added through Apollo program (check the tutorial here), or directly added into the new or modified GFF3 file.

Here's an example:

An updated model slated to replace the reference model, XM_015654027.1:

```
LGIB01000001.1  .       gene    404667  404856  .       -       .       ID=test.gene.1
LGIB01000001.1  .       mRNA    404667  404856  .       -       .       replace=XM_
↪015654027.1;Name=Improved annotation;Parent=test.gene.1;ID=test.mRNA.1
LGIB01000001.1  .       exon    404667  404856  .       -       .       ␣
↪Parent=ID=test.mRNA.1;
LGIB01000001.1  .       CDS     404667  404856  .       -       0       ␣
↪Parent=ID=test.mRNA.1;
```

The reference model to be replaced:

```
LGIB01000001.1  Gnomon  gene    359394  404856  .       -       .       ID=gene28;
LGIB01000001.1  Gnomon  mRNA    359394  404856  .       -       .       ID=rna33;
↪Parent=gene28;Name=XM_015654027.1;
LGIB01000001.1  Gnomon  exon    404667  404856  .       -       .       ID=id260;
↪Parent=rna33;
LGIB01000001.1  Gnomon  exon    362164  362815  .       -       .       ID=id261;
↪Parent=rna33;
LGIB01000001.1  Gnomon  exon    359394  359920  .       -       .       ID=id262;
↪Parent=rna33;
LGIB01000001.1  Gnomon  CDS     404667  404856  .       -       0       ID=cds33;
↪Parent=rna33;
LGIB01000001.1  Gnomon  CDS     362164  362815  .       -       2       ID=cds33;
↪Parent=rna33;
LGIB01000001.1  Gnomon  CDS     359515  359920  .       -       1       ID=cds33;
↪Parent=rna33;
```

6.3.1 Automatically assigning replace tags

(*back*)

You can choose to have the program auto-assign *replace tags* for you. (This is the default behavior.) The program will identify which models from the modified GFF3 file overlap in coding/non-coding sequence with models from the reference GFF3 file. The program will add a 'replace' attribute with the IDs of overlapping models. Specifically, the program will do the following:

- Extract CDS and pre-mRNA sequences from mRNA features from both GFF3 files. (For all other feature types, this program will extract transcript and pre-transcript from both GFF3 files)
- Use blastn to determine which sequences from the modified and reference GFF3 file align to each other **in their coding/non-coding sequence**. These parameters are used: `-evalue 1e-10 -penalty -15 -ungapped`
- If two models pass the alignment step, the program will add a 'replace' attribute with the ID of each overlapping model to the modified gff3 file.
- If no reference model overlaps with a new model, then the program will add 'replace=NA'.
- If one model overlaps another in an intron or UTR (but not within the coding sequence), the auto-assignment program will NOT assign a replace tag. This is because it's not always clear whether the overlapping model should be replaced. You will receive a warning message that this model does not have a replace tag and therefore was not incorporated into the merged gff3 file. You can then go back and manually add a replace tag to the original gff3 file.

6.3.2 Rules for using user-defined files

(*back*)

By default, the program will only use exon to generate spliced sequences for transcripts. If you choose to have the program auto-assign replace tags but there is a model without exon features in your GFF3 files, then you must generate user-defined files for specifying parent and child features for sequences extraction.

Example, a user-defined file for extracting CDS sequences from mRNA, using exon to generate spliced sequences for miRNA and using pseudogenic_exon to generate spliced sequences for pseudogenic_transcript.

User-defined file:

```
mRNA CDS
miRNA exon
pseudogenic_transcript pseudogenic_exon
```

Usage: The user-defined can be specified via **–user_defined_file1** and **–user_defined_file2** argument. You can either give –user_defined_file1 for sequences extraction from updated GFF3 file or give –user_defined_file2 for sequences extraction from reference GFF3 file. Then, the program will use blastn to determine which sequences from the updated and reference GFF3 file align to each other. Specifically, the program will do the blastn with the following query and subject sequences:

- If **–user_defined_file1** is given

Query sequence | Subject sequence — | — user-defined sequences from updated GFF3 file | CDS sequences from reference GFF3 file user-defined sequences from updated GFF3 file | transcript sequences from reference GFF3 file pre-transcript sequences from updated GFF3 file | pre-transcript from reference GFF3 file

- If **–user_defined_file2** is given

Query sequence | Subject sequence — | — CDS sequences from updated GFF3 file | user-defined sequences from reference GFF3 file transcript sequences from updated GFF3 file | user-defined sequences from reference GFF3 file pre-transcript sequences from updated GFF3 file | pre-transcript from reference GFF3 file

- If both **–user_defined_file1** and **–user_defined_file2** are given

Query sequence | Subject sequence — | — user-defined sequences from updated GFF3 file | user-defined sequences from reference GFF3 file pre-transcript sequences from updated GFF3 file | pre-transcript from reference GFF3 file

Note:

- About the parent-child pair, the parent feature should be a transcript (e.g. mRNA, ncRNA) and the child feature is its children (e.g. exon, CDS).
- This program will only generate sequences for the parent-child pair in the user-defined file.

6.3.3 Rules for adding a replace tag on your own

(*back*)

- **If you are replacing non-coding features, and/or replacing coding features with non-coding features, then you must manually include a *replace tag* for these replacement actions.**
- **Replacing a model**: Use the Name or ID attribute of the mRNA or transcript to be replaced. (Don't use the ID or Name of the gene, exon, CDS, or other child features). `replace=CLEC00001-RA`
- **Adding a new model**: Use 'NA' as the replace tag value. `replace=NA`
- **Deleting a reference model**: Use the 'status' attribute with value 'delete' to indicate whether a model from the original gff3 should be deleted. The model that carries the status attribute will NOT be used in the merged gff3. `status=delete`
- **Merging a reference model**: If multiple reference models need to be merged into one, then the modified, merged model should carry replace tags with IDs or Names of all models to be merged. `replace=CLEC00001-RA,CLEC00002-RA`
- **Splitting a reference model**: If a reference model needs to be split, you will need to add a replace tag with the model ID or Name of the split reference model to BOTH models in the modified GFF3. E.g. split model 1: `replace=CLEC00001-RA`, split model 2: `replace=CLEC00001-RA`
- The merge program will check your replace tags, and will throw an error if your replace tag does not meet these assumptions. You will need to update your replace tags according to the error message, and run the program again after fixing.
- If you are using the Apollo manual annotation program at the i5k Workspace to generate the modified GFF3 file, there will be a 'Replaced Models' field in the information editor where you should enter the replace tag information. See https://i5k.nal.usda.gov/apollo-replaced-models-field-explanations-and-examples.

6.4 Replacing and adding models with multiple isoforms

(*back*)

Although the merge program assigns and expects replace tags at the mRNA/transcript level, it essentially behaves as if it should replace models at the gene level. This is not noticeable if both the reference and modified model are single-isoform - however, it may cause confusion with multi-isoform reference models, or if a new isoform should be added. The program assumes that the modified model(s) should have replace tags for ALL isoforms of the gene model to be replaced.

Replacing a multi-isoform model: If a modified model overlaps with a multi-isoform model, the current behavior is to replace ALL isoforms, not single isoforms. The auto-assignment program will assign replace tags corresponding to all overlapping isoforms. The portion of the program that checks the replace tags assumes this behavior. If you added replace tags yourself, and a modified model does not contain replace tags for ALL isoforms of the gene model to be replaced, the program will throw an error, and you will need to add these replace tags for the program to complete.

Adding a new isoform: If you are adding a new isoform to an existing model, you MUST include all reference isoforms that you would like included in the merged GFF3 file to the modified GFF3 file.

Example, one isoform replacing two isoforms. The merged GFF3 file will contain only the single isoform in the modified GFF3 file. The modified GFF3 file contains replace tags for both isoforms of the reference model to be replaced.

Reference GFF3:

```
LGIB01000001.1  Gnomon  gene    1267752 1268637 .       -       .       ID=gene96;
LGIB01000001.1  Gnomon  mRNA    1267752 1268637 .       -       .       ID=rna96;
↪Parent=gene96
LGIB01000001.1  Gnomon  exon    1268346 1268637 .       -       .       Parent=rna96
LGIB01000001.1  Gnomon  exon    1267752 1268263 .       -       .       Parent=rna96
LGIB01000001.1  Gnomon  CDS     1268346 1268637 .       -       0       Parent=rna96
LGIB01000001.1  Gnomon  CDS     1267818 1268263 .       -       2       Parent=rna96

LGIB01000001.1  Gnomon  gene    1267818 1268637 .       -       .       ID=gene100
LGIB01000001.1  Gnomon  mRNA    1267818 1268637 .       -       .       ID=rna100;
↪Parent=gene100
LGIB01000001.1  Gnomon  exon    1267818 1268263 .       -       .       Parent=rna100
LGIB01000001.1  Gnomon  exon    1268346 1268637 .       -       .       Parent=rna100
LGIB01000001.1  Gnomon  CDS     1267818 1268263 .       -       2       Parent=rna100
LGIB01000001.1  Gnomon  CDS     1268346 1268637 .       -       0       Parent=rna100
```

Modified GFF3:

```
LGIB01000001.1  .       gene    1267752 1268263 .       -       .       ID=geneID1;
LGIB01000001.1  .       mRNA    1267752 1268263 .       -       .       ␣
↪Parent=geneID1;ID=mrnaID1;replace=rna96,rna100
LGIB01000001.1  .       exon    1267752 1268263 .       -       .       ␣
↪Parent=mrnaID1;
LGIB01000001.1  .       CDS     1267818 1268261 .       -       0       ␣
↪Parent=mrnaID1;
```

Example, adding a new isoform. The merged GFF3 file will contain all information from the modified GFF3 file. The modified GFF3 file contains both isoforms, even though one of the isoforms has identical coordinates to the reference isoform. Both mRNAs in the modified GFF3 file contain the same replace tags, because they both replace the reference model rna96.

Reference GFF3:

```
LGIB01000001.1  Gnomon  gene    1267752 1268637 .       -       .       ID=gene96;
LGIB01000001.1  Gnomon  mRNA    1267752 1268637 .       -       .       ID=rna96;
↪Parent=gene96
LGIB01000001.1  Gnomon  exon    1268346 1268637 .       -       .       Parent=rna96
LGIB01000001.1  Gnomon  exon    1267752 1268263 .       -       .       Parent=rna96
LGIB01000001.1  Gnomon  CDS     1268346 1268637 .       -       0       Parent=rna96
LGIB01000001.1  Gnomon  CDS     1267818 1268263 .       -       2       Parent=rna96
```

Modified GFF3:

```
LGIB01000001.1  .       gene    1267752 1268637 .       -       .       ID=geneID1
LGIB01000001.1  .       mRNA    1267752 1268263 .       -       .       ␣
↪Parent=geneID1;ID=mRNAID1;replace=rna96
LGIB01000001.1  .       exon    1267752 1268263 .       -       .       Parent=mRNAID1
LGIB01000001.1  .       CDS     1267818 1268261 .       -       0       Parent=mRNAID1
LGIB01000001.1  .       mRNA    1267752 1268637 .       -       .       ␣
↪Parent=geneID1;ID=mRNAID2;replace=rna96
```

(continues on next page)

(continued from previous page)

```
LGIB01000001.1  .       exon    1268346 1268637 .       -       .       Parent=mRNAID2
LGIB01000001.1  .       CDS     1268346 1268637 .       -       0       Parent=mRNAID2
LGIB01000001.1  .       CDS     1267818 1268263 .       -       2       Parent=mRNAID2
LGIB01000001.1  .       exon    1267752 1268263 .       -       .       Parent=mRNAID2
```

6.5 Odd use cases

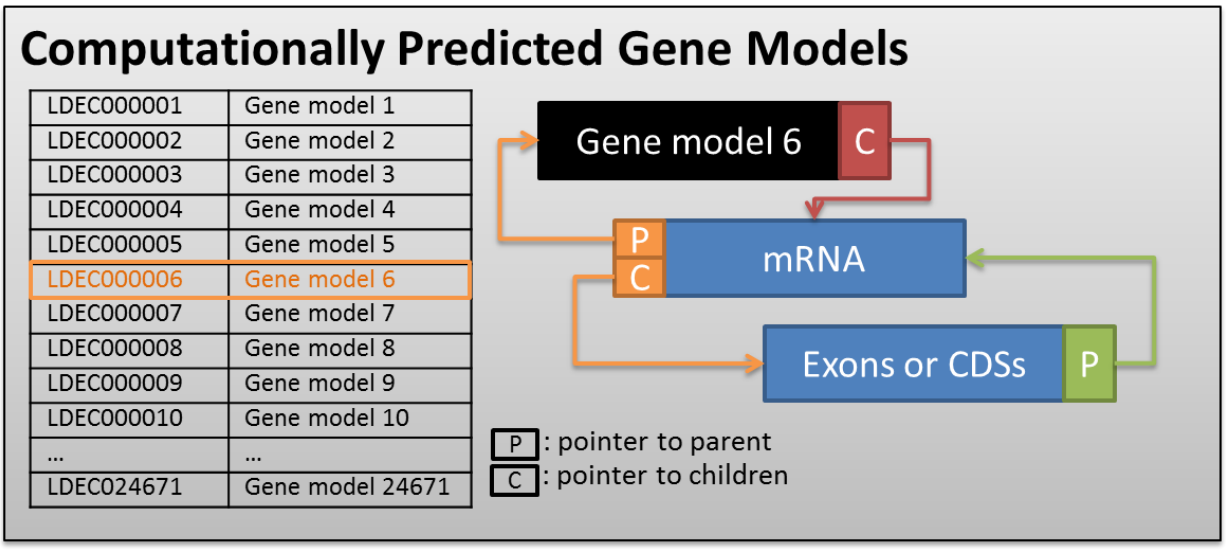
(*back*)

- If you are replacing non-coding features, and/or replacing coding features with non-coding features, then you must manually include a replace tag for these replacement actions.
- It is possible for a modified model to have multiple isoforms that do not share CDS with each other - for example with partial models due to a poor genome assembly. In this case, the auto-assignment program will assign different replace tags to each isoform, but will then reject these auto-assigned replace tags because it expects isoforms of a gene model to have the same replace tags (see section "*Some notes on multi-isoform models*", above). You'll need to add the replace tags manually - all isoforms should carry the replace tags of all models to be replaced by the whole gene model.
- If one model overlaps another in an intron or UTR (but not within the coding sequence), the auto-assignment program will NOT assign a replace tag. This is because it's not always clear whether the overlapping model should be replaced. You will receive a warning message that this model does not have a replace tag and therefore was not incorporated into the merged gff3 file. You can then go back and manually add a replace tag to the original gff3 file.
- Note that gff3_merge will NOT replace the ID attributes for existing features in your gff3 files. Therefore, if a new feature is added into the merged file that has an identical ID with an existing feature, then there will be duplicate IDs for this feature in the merged gff3 file.

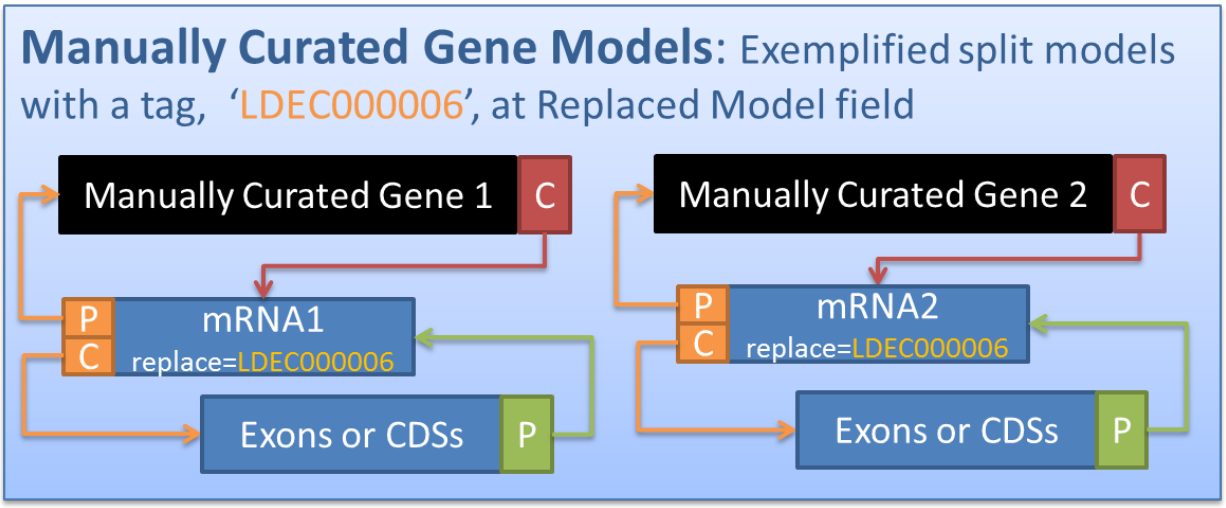
6.6 How the merge works

(*back*)

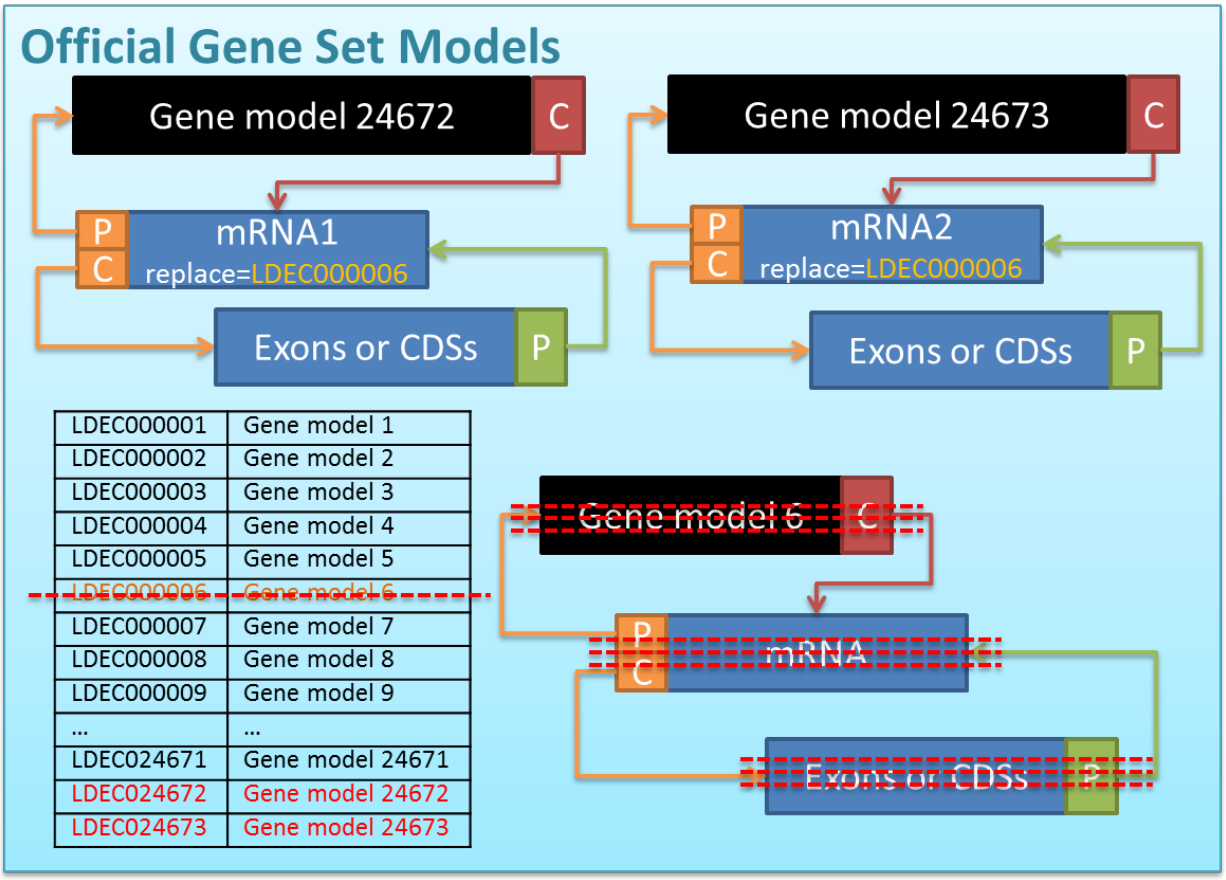
In this pipeline, a GFF3 file is parsed into a structure composed of simple python dict and list. Within a list, every gene model uses a tree structure to store the relationships between parents and children. The figure below showed an example (LDEC000006) how it works on the gff file of computationally predicted gene models.



The same structure is also applied on the gff file of manually curated gene models, but one more attribute, 'replace=', is added and required for manually curated gene models. This mandatory 'Replaced Models' field specifies which gene models from the computationally predicted gene set should be replaced by the manually curated models. Here we provide an example that a computationally predicted gene model (LDEC000006) is split into two models after manual curation (See the figure below), and thus both of the manually curated models has the replace tag, LDEC000006.



During the MERGE phase, the LDEC000006 is removed from the python list/dict of computationally predicted gene set, as well as the tree structure of LDEC000006. Then, the two manually curated models are added into the python list/dict, and assigned with new unique IDs because this is a split replacement. Again, model IDs are handled as follows: For simple replacement, the IDs are inherited from the replaced computationally predicted models. For other types of replacement, new unique IDs are assigned. (*back to the top*)


Official Gene Set Models
Gene model 24672
C
P
C
mRNA1
replace=LDEC000006
Exons or CDSs
P
Gene model 24673
C
P
C
mRNA2
replace=LDEC000006
Exons or CDSs
P
LDEC000001 Gene model 1
LDEC000002 Gene model 2
LDEC000003 Gene model 3
LDEC000004 Gene model 4
LDEC000005 Gene model 5
LDEC000006 Gene model 6
LDEC000007 Gene model 7
LDEC000008 Gene model 8
LDEC000009 Gene model 9
... ...
LDEC024671 Gene model 24671
LDEC024672 Gene model 24672
LDEC024673 Gene model 24673
Gene model 6
C
P
C
mRNA
Exons or CDSs
P

CHAPTER 7

gff3_sort readme

Sort features in a gff3 file by according to their order on a scaffold, their coordinates on a scaffold, and parent-child relationships.

7.1 Inputs:

1. GFF3 file: Specify the file name with the -g argument

7.2 Outputs:

1. Sorted GFF3 file: Specify the file name with the -og argument
    - All related features (with parent-child relationships) are separated by ### directives for easier downstream parsing

7.3 Usage:

1. Specify the input, output file names and options using short arguments:
    - `gff3_sort -g example_file/example.gff3 -og example_file/example_sorted.gff`
2. Specify the input, output file names and options using long arguments:
    - `gff3_sort --gff_file example_file/example.gff3 --output_gff example_file/example_sorted.gff`

7.4 Optional arguments:

1. -h, –help
   - show this help message and exit
2. -g GFF_FILE, –gff_file GFF_FILE
   - GFF3 file that you would like to sort.
3. -og OUTPUT_GFF, –output_gff OUTPUT_GFF
   - Sorted GFF3 file
4. -t, SORT_TEMPLATE, –sort_template SORT_TEMPLATE
   - A file that indicates the sorting order of features within a gene model
5. -i, –isoform_sort
   - Sort multi-isoform gene models by feature type (default: False)
6. -v, –version
   - show program's version number and exit

7.5 Example:

7.5.1 Sort gff3 file without a sort template file

- example command:

```
gff3_sort --gff_file example.gff3 --output_gff example_sort.gff3
```

- Input gff3 file:

```
LGIB01000001.1  Gnomon  gene    52056   58768   .       +       .       ID=gene1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna1;
↪Parent=gene1
LGIB01000001.1  Gnomon  CDS     52056   52096   .       +       0       ID=cds1;
↪Parent=rna1
LGIB01000001.1  Gnomon  exon    52056   52096   .       +       .       ID=id4;
↪Parent=rna1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna2;
↪Parent=gene1
LGIB01000001.1  Gnomon  CDS     52100   53000   .       +       0       ID=cds2;
↪Parent=rna2
LGIB01000001.1  Gnomon  exon    52056   53000   .       +       .       ID=id19;
↪Parent=rna2
```

- Output gff3 file:

```
LGIB01000001.1  Gnomon  gene    52056   58768   .       +       .       ID=gene1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna1;
↪Parent=gene1
LGIB01000001.1  Gnomon  exon    52056   52096   .       +       .       ID=id4;
↪Parent=rna1
LGIB01000001.1  Gnomon  CDS     52056   52096   .       +       0       ID=cds1;
↪Parent=rna1
```

(continues on next page)

(continued from previous page)

```
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna2;
↪Parent=gene1
LGIB01000001.1  Gnomon  exon    52056   53000   .       +       .       ID=id19;
↪Parent=rna2
LGIB01000001.1  Gnomon  CDS     52100   53000   .       +       0       ID=cds2;
↪Parent=rna2
```

7.5.2 Sort gff3 file with a sort template file

- sort template file: A file that indicates the sorting order of features within a gene model. Feature type with the same sorting order should be in the same line and split by space.

```
gene pseudogene
mRNA
exon
CDS
```

Sort gff3 file without –isoform_sort

- example command:

```
gff3_sort --gff_file example.gff3 --sort_template sort_template.txt
--output_gff example_sort.gff3
```

- Output gff3 file:

```
LGIB01000001.1  Gnomon  gene    52056   58768   .       +       .       ID=gene1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna1;
↪Parent=gene1
LGIB01000001.1  Gnomon  exon    52056   52096   .       +       .       ID=id4;
↪Parent=rna1
LGIB01000001.1  Gnomon  CDS     52056   52096   .       +       0       ID=cds1;
↪Parent=rna1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna2;
↪Parent=gene1
LGIB01000001.1  Gnomon  exon    52056   53000   .       +       .       ID=id19;
↪Parent=rna2
LGIB01000001.1  Gnomon  CDS     52100   53000   .       +       0       ID=cds2;
↪Parent=rna2
```

Note:

If not all the feature type are documented in the sort template file. gff3_sort will sort features by level(1st-level, 2nd-level, and etc) and then by the order in sort template file.

- sort template file:

```
gene pseudogene
CDS
```

- Output gff3 file:

```
LGIB01000001.1  Gnomon  gene    52056   58768   .       +       .       ID=gene1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna1;
↪Parent=gene1
```

(continues on next page)

(continued from previous page)

```
LGIB01000001.1  Gnomon  CDS     52056   52096   .       +       0       ID=cds1;
↪Parent=rna1
LGIB01000001.1  Gnomon  exon    52056   52096   .       +       .       ID=id4;
↪Parent=rna1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna2;
↪Parent=gene1
LGIB01000001.1  Gnomon  CDS     52100   53000   .       +       0       ID=cds2;
↪Parent=rna2
LGIB01000001.1  Gnomon  exon    52056   53000   .       +       .       ID=id19;
↪Parent=rna2
```

Sort gff3 file with –isoform_sort

- example command:

```
gff3_sort --gff_file example.gff3 --sort_template sort_template.txt
--isoform_sort --output_gff example_sort.gff3
```

- Output gff3 file:

```
LGIB01000001.1  Gnomon  gene    52056   58768   .       +       .       ID=gene1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna1;
↪Parent=gene1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna2;
↪Parent=gene1
LGIB01000001.1  Gnomon  exon    52056   53000   .       +       .       ID=id19;
↪Parent=rna2
LGIB01000001.1  Gnomon  exon    52056   52096   .       +       .       ID=id4;
↪Parent=rna1
LGIB01000001.1  Gnomon  CDS     52056   52096   .       +       0       ID=cds1;
↪Parent=rna1
LGIB01000001.1  Gnomon  CDS     52100   53000   .       +       0       ID=cds2;
↪Parent=rna2
```

Note:

If not all the feature type are documented in the sort template file. gff3_sort will sort features by the order in sort template file and then by level(1st-level, 2nd-level, and etc).

- sort template file:

```
gene pseudogene
CDS
```

- Output gff3 file:

```
LGIB01000001.1  Gnomon  gene    52056   58768   .       +       .       ID=gene1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna1;
↪Parent=gene1
LGIB01000001.1  Gnomon  CDS     52056   52096   .       +       0       ID=cds1;
↪Parent=rna1
LGIB01000001.1  Gnomon  exon    52056   52096   .       +       .       ID=id4;
↪Parent=rna1
LGIB01000001.1  Gnomon  mRNA    52056   58768   .       +       .       ID=rna2;
↪Parent=gene1
LGIB01000001.1  Gnomon  CDS     52100   53000   .       +       0       ID=cds2;
↪Parent=rna2
```

(continues on next page)

(continued from previous page)

```
LGIB01000001.1  Gnomon  exon    52056   53000   .       +       .       ID=id19;
↪Parent=rna2
```

7.6 Assumptions:

1. Any features without a Parent attribute are 'root' features - the program will insert directives (lines beginning with ##) above these features.
2. All child features occur after their respective Parent feature, but before new Parent features.

CHAPTER 8

gff3_to_fasta readme

Extract sequences from specific regions of genome based on gff file.

8.1 Features

- **Incorporation of gff3.py**: `gff3.py` is contributed by Han Lin which uses simple data structures to parse a [GFF3] file into a structure composed of simple python [dict] and [list].
- **Validation**: Validate the GFF3 formatting errors utilizing QC methods contributed by the I5K Workspace@NAL team. Provide WARNING messages for gene models that may have incorrect biological sequences generated because of [GFF3] formatting errors.
- **Easy extraction of biological sequences**: Provide options for extracting six types of biological sequences or user-specified type of spliced sequences.
  - **gene**: Gene sequence for each record in the [FASTA] output. Gene or pseudogene features need to be included in the gff file
  - **exon**: Exon sequence for each record in the [FASTA] output. Exon features need to be included in the gff file
  - **pre_trans**: Genomic region of a transcript model, namely premature transcript (exon and intron regions included), for each record in the [FASTA] output. Transcript-level features (such as mRNA, rRNA, pseudogenic transcripts) need to be included in the gff file.
  - **trans**: Spliced transcript (only exons included) for each record in the [FASTA] output. Exon features are mainly used for splicing. CDS features are used instead if exon features are absent. If both cds and exon features are absent, the transcript is not generated and a WARNING message is shown with the transcript ID.
  - **cds**: Coding sequence (utr exons and introns excluded) for each record in the [FASTA] output. CDS features need to be included in the gff file.
  - **pep**: Translated peptide sequences (translation based on cds regions) for each record in the [FASTA] output. CDS features need to be included in the gff file.

  - **`user_defined`**: Specify parent and child features for fasta extraction via the -u argument, format [parent feature type] [child feature type].(e.g. `-st user_defined -u miRNA exon`)

- **`translator` method for universal translation**: The `translator` method is feasible for
  - translation from 64 combitions of standard codons (Only standard codons and universal stop condons are considered.)
  - translation from codons with IUB Depiction
  - translation from mRNA (U contained) or CDS (T, instead of U contained)

8.2 Usage

gff3_to_fasta.py [-h] [-g GFF] [-f FASTA] [-st SEQUENCE_TYPE] [-u USER_DEFINED] [-d DEFLINE] [-o OUTPUT_PREFIX] [-noQC] [-v]

8.3 Testing enviroment

1. Python 2.7

8.4 Required inputs

1. GFF3: specify the file name with the -g argument
2. Fasta file: specify the file name with the -f argument. This file **must** be the Fasta file that the GFF3 seqids and coordinates refer to. For more information, refer to the GFF3 specification.
3. Output prefix: specify with the -o argument. All resulting fasta files will contain this prefix.

8.5 Outputs

1. Fasta formatted sequence file based on the gff3 file.

8.6 Example command

1. Specify the input, output file names and options using short arguments:
   - `gff3_to_fasta -g example_file/example.gff3 -f example_file/reference.fa -st all -d simple -o test_sequences`

8.7 Optional arguments

1. -h, –help
   - show this help message and exit
2. -g GFF, –gff GFF

- Genome annotation file in GFF3 format

3. -f FASTA, –fasta FASTA
   - Genome sequences in FASTA format
4. -embf, –embedded_fasta
   - Specify this option if you want to extract sequence from embedded fasta.
5. -st SEQUENCE_TYPE, –sequence_type SEQUENCE_TYPE
   - Type of sequences you would like to extract:
     - “all” - FASTA files for all types of sequences listed below, except user_defined;
     - “gene” - gene sequence for each record;
     - “exon” - exon sequence for each record;
     - “pre_trans” - genomic region of a transcript model (premature transcript);
     - “trans” - spliced transcripts (only exons included);
     - “cds” - coding sequences;
     - “pep” - peptide sequences;
     - “user_defined” - specify parent and child features via the -u argument.
6. -u USER_DEFINED, –user_defined USER_DEFINED
   - Specify parent and child features for fasta extraction, format [parent feature type] [child feature type]. Required if -st user_defined is given.
     - Example: -st user_defined -u miRNA exon
   - Lines with the child feature type given in -u must contain a Parent attribute referencing the given Parent feature type. Hence, the parent lines must also contain an ID attribute.
   - If CDS is the child feature type, the program will take phase into account.
7. -d DEFLINE, –defline DEFLINE
   - Defline format in the output FASTA file:
     - “simple” - only ID is shown in the defline;
     - “complete” - complete information of the feature is shown in the defline.
8. -o OUTPUT_PREFIX, –output_prefix OUTPUT_PREFIX
   - Prefix of output file name
9. -noQC, –quality_control
   - Specify this option if you do not want to excute quality control for gff file. (default: QC is executed)
10. -v, –version
    - Show program version number and exit

CHAPTER 9

FAQ

9.1 Q: When installing, the program fails with following message: `ImportError: No module named wheel.bdist_wheel.`

Since 1.4.2, we use wheel to build our python package. This error message means that you don't have wheel on your machine. Use `pip install wheel` to install it first.

9.2 Q: When running one of the GFF3-toolkit programs, the program fails with a stack trace error.

Usually, this means that there is a problem with the input file. We are working on having each program output error messages with the input file line number. In the meantime, send us your input file and we can help figure out what the problem is.

9.3 Q: What are the licensing terms for this project?

This software/database is a "United States Government Work" under the terms of the United States Copyright Act. It was written as part of the author's official duties as a United States Government employee and thus cannot be copyrighted. This software/database is freely available to the public for use. The National Agriculture Library and the U.S. Government have not placed any restriction on its use or reproduction. (Please see LICENSE.md)

9.4 Q: What kind of errors can be detected by gff3_QC.py? (Detection of GFF3 format errors: gff3_QC.py)

Currently, ~50 types of formatting errors can be detected. Errors are detected by reviewing three types of feature sets in a GFF3 file, and thus are grouped into three categories (Error category – feature type):

- Intra-model errors (Ema) – multiple features within a model
- Inter-model errors (Emr) – multiple features across models
- Single feature errors (Esf) – each single feature.

Please view the full documentation of gff3_QC.py for the full list of detected error types.

9.5 Q: Why is gff3_QC.py taking so long to run? (Detection of GFF3 format errors: gff3_QC.py)

gff3_QC.py can take a while if your gff3 file is large - please be patient!

9.6 Q: Why does the sorted gff3 file have a different number of lines than the input file? (Sort a GFF3 file: gff3_sort.py)

The program gff3_sort.py automatically ignores the hash tag lines other than ##gff-version 3 and ### while sorting a GFF3 file. After sorting, the program puts one line of ### between every gene model in the output GFF3. Therefore, the total lines of the output file might be different from the input. To check the consistency of the lines, please use the following command,

grep -v "#" input.gff |wc -l

grep -v "#" sorted.gff |wc -l

In addition, if your input gff file contains a feature that has two or more parent IDs, the program replicates the feature and lists it under each parent. Thus, the output file would have more lines than the input file.

9.7 Q: Which codons are considered for translation? (Generate biological sequences from a GFF3 file: gff3_to_fasta.py)

Translation from 64 combinations of standard codons (Only standard codons and universal stop codons are considered.)

9.8 Q: Why does gff3_merge.py sometimes reject auto-assigned replace tags when the reference model has multiple isoforms? (Merge 2 GFF3 files: gff3_merge.py)

It is possible for a modified model to have multiple isoforms that do not share CDS with each other - for example with partial models due to a poor genome assembly. In this case, the auto-assignment program will assign different replace tags to each isoform, but will then reject these auto-assigned replace tags because it expects isoforms of a gene model to have the same replace tags (see section "Some notes on multi-isoform models", above). You'll need to add the replace tags manually - all isoforms should carry the replace tags of all models to be replaced by the whole gene model.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search